Adversarial Robustness of Multigraph Neural Networks

Master Thesis

Dennis Heijmans

Delft University of Technology

TU Delft

Adversarial Robustness of Multigraph Neural Networks

by

Dennis Heijmans

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday, July 2, 2026 at 3:45 PM.

Student number:	5444993	
Project duration:	November 10, 2025 – July 2, 2026	
Thesis committee:	Dr. ir. J. A. Pouwelse,	TU Delft, Chair
	Dr. Z. Erkin,	TU Delft, Core Member
	Dr. K. Atasu,	TU Delft, Core Member
	H. Ç. Bilgi,	TU Delft, Advisor

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

Preface

This thesis marks the conclusion of my studies at TU Delft. About a year ago, a course on Scalable Learning Systems introduced me to graph machine learning, a subject I found myself drawn to immediately, and it was the contact it brought that opened the door to adversarial robustness on graph-structured data. What deepened my investment even more was the real-world stakes behind the technical problem: money laundering, sustaining organised crime and corruption on a global scale.

I owe this path to my supervisor Dr. Kubilay Atasu. I felt he believed in my potential already from early on, and that assured me to explore and develop my own approach. That he now sees the value in the work and pursues publication at a conference means a lot to me.

I am equally grateful to my co-supervisors Çağrı Bilgi and Dr. Rui Wang. Çağrı offered a great deal of practical guidance and ideas that helped the project along, and our close collaboration was something I valued deeply. Rui helped me with my writing and helped me be more precise in my argumentation.

I would also like to thank Dr. Zeki Erkin and Dr. Johan Pouwelse for their interest in this work and for their willingness to serve on my thesis committee.

Lastly, I am grateful to my family and friends for their support during what was a turbulent period, in which I also had to come to terms with a type 1 diabetes diagnosis. Their support made it possible to persevere.

Dennis Heijmans
Roosendaal, June 2026

Abstract

Detecting money laundering in financial transaction data is a task where graph neural networks (GNNs) have shown strong potential. Such data is naturally represented as a directed multigraph, since two accounts, each represented as a node, may exchange many separate payments, each forming a distinct edge with its own amount, currency, and timestamp. Preserving these parallel edges, rather than collapsing them into a single connection, retains the fine-grained structure that allows for distinguishing laundering behaviour from ordinary activity. Yet these models also introduce a new vulnerability, as an adversary could manipulate the transaction graph to alter the neighbourhood of a suspicious account such that the GNN misclassifies it as benign. Existing adversarial robustness research operates on the adjacency matrix, which records at most one edge per node pair and therefore cannot represent the parallel transactions between two accounts that this task depends on. Multigraph GNNs therefore lack both a framework for evaluating robustness under structural perturbations and defences against such perturbations.

This thesis extends adversarial robustness analysis to multigraph GNNs through three contributions. First, it reformulates GNN message passing and attack optimisation over the incidence matrix instead of the adjacency matrix, yielding the first gradient-based structural attack that retains multi-edge structure. Second, it introduces unnoticeability loss terms that constrain perturbations to maintain the graph's statistical fingerprint, including the frequency of characteristic patterns such as short transaction cycles, keeping the attack statistically plausible and unnoticeable at the macro level. Third, it scales the framework to large networks with projected randomised block coordinate descent. On the IBM synthetic anti-money laundering dataset, learned attacks substantially reduce detection accuracy compared to non-learnable perturbations, and adversarial training recovers robustness, showing that multigraph GNNs are both vulnerable to structural manipulation and defensible against it.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AML	Anti-money laundering
GIN	Graph isomorphism network
GNN	Graph neural network
MLP	Multilayer perceptron
PGD	Projected gradient descent
PR-BCD	Projected randomised block coordinate descent
ReLU	Rectified linear unit

Symbols

Symbol	Definition
$\boldsymbol{A}$	Adjacency matrix
$\bar{\boldsymbol{A}}$	Complement graph ($\mathbf{1}\mathbf{1}^\top - \boldsymbol{I} - \boldsymbol{A}$)
$\boldsymbol{b}^{(\ell)}$	Bias vector of layer ℓ
b	PR-BCD block size
$\mathcal{B}^{(t)}$	Random edge block sampled at iteration t
$\boldsymbol{C}$	Edge flip direction matrix ($\bar{\boldsymbol{A}} - \boldsymbol{A}$)
d	Node feature dimension
$\mathcal{E}$	Edge set
f_θ	GNN model with parameters θ
$\mathcal{G}$	Graph
$\boldsymbol{h}_v^{(k)}$	Representation of node v at layer k
K	Number of message-passing (GNN) layers
L	Number of MLP layers
$\mathcal{L}_{\text{cls}}$	Classification loss
$\mathcal{L}_{\text{attack}}$	Attack loss (low when GNN misclassifies many nodes)
n	Number of nodes
$\mathcal{N}(v)$	Neighbourhood of node v
$\boldsymbol{S}$	Perturbation indicator matrix ($\boldsymbol{S}_{ij} = 1$ if edge (i, j) is flipped)
$\mathcal{N}_S(v)$	Sampled neighbourhood of node v
S	Neighbourhood sampling fan-out
$\mathcal{P}$	Feasible set of perturbations
T	Number of inner attack steps
$\boldsymbol{W}^{(\ell)}$	Weight matrix of layer ℓ
$\boldsymbol{X}$	Node feature matrix
$\boldsymbol{x}$	Input feature vector
$\boldsymbol{y}$	Ground-truth labels
$\hat{\boldsymbol{y}}$	Model prediction
α	Attack step size
Δ	Perturbation budget (number of edges)
$\epsilon^{(k)}$	GIN learnable scalar at layer k
ε	Adversarial perturbation magnitude

Symbol	Definition
η	Learning rate
θ	Model parameters
σ	Non-linear activation function

1

Introduction

Financial institutions are legally required to detect money laundering, the practice of moving the proceeds of illegal activity through sequences of transactions to disguise their origin. Estimates place the annual volume of laundered funds between 2% and 5% of global GDP, equivalent to hundreds of billions of US dollars [16]. Transaction volumes make manual review of suspicious activity infeasible [1], and automated detection systems are therefore essential for regulatory compliance and financial security.

Financial transaction data is naturally graph-structured, where accounts can be represented as nodes and each individual transaction as an edge between the two accounts involved. The same pair of accounts may exchange many separate payments over time, so transaction graphs are *multigraphs*, graphs in which multiple edges can connect the same pair of nodes. This structure matters for detection because a simple graph, which allows at most one edge per node pair, collapses all transactions between two accounts into a single connection, discarding individual amounts, timestamps, and currencies. A multigraph retains each transaction as a distinct edge, preserving the parallel-edge structure, along with the per-transaction attributes, that a collapsed representation would discard [5].

Graph neural networks (GNNs) are machine learning models that learn a representation for each node by aggregating information from its graph neighbours [13, 17, 20, 10]. Applied to transaction data, a GNN assigns each account a feature vector encoding not only the account's own attributes but also the transactional context of its neighbourhood, capturing structural signatures of laundering behaviour such as smurfing and circular fund flows [11]. Recently, specialised GNN architectures have been developed for financial transaction multigraphs [3, 4] and demonstrate higher F1 scores than standard GNNs on anti-money laundering (AML) detection tasks [5, 1].

Despite this strong performance, GNN-based detection systems introduce a new risk. An adversary who can probe the model may manipulate the transaction graph to evade detection. By redirecting payments to different counterparties, the adversary can alter the neighbourhood of a suspicious account so that the GNN assigns it a benign label. This manipulation strategy, an adversarial structural perturbation, has been shown to degrade GNN accuracy substantially even when the number of modified edges is small [19, 21, 7, 9]. In the AML context, a successful perturbation could allow laundering activity to pass undetected at scale.

Prior adversarial robustness research on GNNs operates on the adjacency matrix [19, 21, 7, 9, 12], which records at most one edge per node pair and therefore cannot represent the parallel edges between the same node pair that characterise a multigraph. Consequently, the robustness of multigraph GNNs remains unexplored, with no attacks that operate on the multigraph's structure and no defences against them.

This thesis extends adversarial robustness analysis to multigraph GNNs. The core contribution is an adversarial robustness framework for multigraph GNNs, presented as a scientific paper in Chapter 3. Three contributions make up the framework:

1. We develop the first gradient-based structural attack method for multigraph GNNs, reformulating

GNN message passing and attack optimisation using the incidence matrix rather than the adjacency matrix, enabling gradient-based optimisation over a continuously relaxed incidence matrix that retains multi-edge structure.

2. We introduce unnoticeability loss terms that constrain adversarial perturbations to preserve the multigraph's macro-level statistical fingerprint, in particular the frequency of short transaction cycles. Adversarial attacks must be unnoticeable to be effective, since a defender who spots the perturbation can neutralise the attack without engaging the GNN at all [12]. In the graph domain, unnoticeability requires numerical measures. Cycle-count statistics serve as such a measure, keeping the attack statistically plausible and undetectable at the macro level.
3. We scale the framework to large financial networks using projected randomised block coordinate descent (PR-BCD) adapted to the incidence-matrix setting, and demonstrate that adversarial training can harden multigraph GNNs against the proposed structural attacks.

All three contributions operate on graph structure alone: the attack rewires transaction targets while holding node and edge features fixed.

The broader importance of this work lies in its direct connection to financial integrity. Money laundering enables and sustains a wide range of serious crimes by obscuring the origin of illicit funds. Detection systems that can be bypassed by a knowledgeable adversary provide a false sense of security, allowing laundering activity to continue at scale. Studying attacks on GNN-based AML systems is therefore not an end in itself, but the necessary first step toward building detection systems whose robustness can be tested and verified. The adversarial training results in this thesis show that multigraph GNNs can be hardened against structural attacks, pointing toward systems that remain reliable even under active adversarial pressure.

Chapter 2 builds the technical background required to follow the scientific paper. It begins with the fundamentals of deep learning and neural networks, then introduces graph neural networks and builds up to the Graph Isomorphism Network (GIN), the architecture evaluated throughout the paper. The chapter closes with adversarial robustness on graphs, covering the threat model, the projected gradient descent (PGD) and PR-BCD attack methods, and adversarial training as a defence. Chapter 3 presents the scientific paper that develops and evaluates the adversarial robustness framework for multigraph GNNs, the core contribution of this thesis.

2

Background

This chapter builds the technical foundation needed to follow Chapter 3. It covers three areas in sequence. The first section introduces neural networks and the gradient-based optimisation used to train them. The second builds up the graph neural network framework, from basic graph definitions to the Graph Isomorphism Network (GIN). The third covers adversarial robustness of graph neural networks, introducing the threat model that defines how an attacker may act and the gradient-based methods used to find structural perturbations on simple graphs, closing with adversarial training as a strategy for hardening models against such perturbations. A final section presents the money laundering patterns that the detection task in Chapter 3 targets, connecting the technical machinery to the financial problem it serves.

2.1. Deep Learning & Neural Networks

A neural network is built from learned layers stacked in sequence. This section introduces the multilayer perceptron and the gradient-based optimisation used to train it.

2.1.1. Neural Networks and Multilayer Perceptrons

A neural network is a function composed of multiple layers, each applying a learned linear transformation followed by a fixed non-linear activation. Given an input vector $\boldsymbol{x} \in \mathbb{R}^d$, a single layer produces an output

$$\boldsymbol{h} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.1}$$

where $\boldsymbol{W} \in \mathbb{R}^{d' \times d}$ is a weight matrix, $\boldsymbol{b} \in \mathbb{R}^{d'}$ is a bias vector, and σ is a non-linearity applied element-wise. One common choice of non-linearity is the *rectified linear unit* $\mathrm{ReLU}(z) = \max(0, z)$, the activation used throughout Chapter 3. A *multilayer perceptron* (MLP) stacks L such layers in sequence, with the output of each layer serving as the input to the next:

$$\boldsymbol{h}^{(0)} = \boldsymbol{x}, \qquad \boldsymbol{h}^{(\ell)} = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\right), \quad \ell = 1, \ldots, L. \tag{2.2}$$

The final output $\boldsymbol{h}^{(L)}$ is the network's prediction. The weight matrices and biases are collectively the model's *parameters* θ, which are learned from data as described in Section 2.1.2. In Chapter 3, an MLP appears inside each layer of GIN, transforming aggregated node representations into updated ones (Section 2.2.5).

2.1.2. Training

A network's parameters are adjusted to minimise a *loss function* $\mathcal{L}_{\text{cls}}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ that measures the discrepancy between the model's predictions $\hat{\boldsymbol{y}}$ and the ground-truth labels $\boldsymbol{y}$. A common choice for binary prediction tasks is *binary cross-entropy* (BCE). For a predicted probability $\hat{p} \in [0, 1]$ and a true label $y \in \{0, 1\}$,

$$\mathrm{BCE}(y, \hat{p}) = -[y \log \hat{p} + (1 - y) \log(1 - \hat{p})], \tag{2.3}$$

which is low when $\hat{p}$ is close to y and high otherwise. For multi-label classification, where each node carries several independent binary labels, the total loss sums BCE over all labels and all labelled nodes.

The standard optimisation method is *gradient descent*. Starting from an initial parameter vector, each update step moves θ in the direction that most reduces the loss,

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{cls}}, \tag{2.4}$$

where $\eta > 0$ is the *learning rate* (step size) and $\nabla_\theta \mathcal{L}_{\text{cls}}$ is the gradient of the loss with respect to θ, computed via backpropagation. In practice, gradients are estimated on small random subsets of training examples, a variant known as *stochastic gradient descent*.

This same mechanism underpins both model training and adversarial attacks. In both cases, gradient-based optimisation adjusts variables to minimise a loss, differing only in what is being optimised.

2.2. Graph Neural Networks

Graph neural networks learn representations over graph-structured data by propagating information along edges. This section builds from the basic definition of a graph to the message passing framework common to most GNN architectures, and then to the Graph Isomorphism Network, a specific architecture with strong theoretical expressiveness.

2.2.1. Graphs

A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a node set $\mathcal{V} = \{v_1, \ldots, v_n\}$ of size n and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The graph's topology is encoded by the *adjacency matrix* $\boldsymbol{A} \in \{0,1\}^{n \times n}$, where $\boldsymbol{A}_{ij} = 1$ if an edge from v_i to v_j exists and $\boldsymbol{A}_{ij} = 0$ otherwise. Each node carries a feature vector, collected into a *node feature matrix* $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where row i is the initial representation $\boldsymbol{h}_i^{(0)} \in \mathbb{R}^d$ of node v_i. The set of nodes directly connected to a node v_i is its *neighbourhood* $\mathcal{N}(v_i)$.

These definitions apply to *simple graphs*, in which at most one edge connects each ordered node pair. The article in Chapter 3 operates on *multigraphs*: graphs in which multiple distinct edges may connect the same pair of nodes. In the financial transaction setting, each edge represents one individual payment, so parallel edges between the same account pair carry transaction-level information, such as amounts, timestamps, and frequencies, that a single adjacency-matrix entry cannot preserve. Figure 2.1 illustrates this distinction.

A multigraph is instead represented by its *incidence matrix* $\boldsymbol{B} \in \mathbb{R}^{n \times m}$, with one row per node and one column per edge. Each column marks the source and target of a single edge, so parallel edges between the same node pair occupy separate columns and stay distinct, where a single adjacency entry would merge them. Because every individual edge is addressable, a structural perturbation can be written as a continuous, differentiable change to $\boldsymbol{B}$. Standard GNN attacks operate directly on the adjacency matrix $\boldsymbol{A}$ and so cannot be applied to the per-edge multigraph formulation; Chapter 3 builds its attack on the incidence matrix to address this incompatibility.

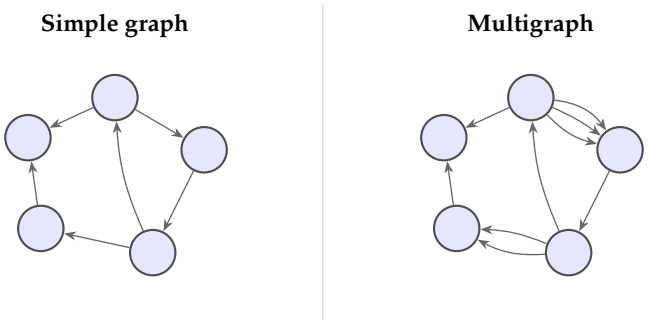

Figure 2.1: Simple graph vs. multigraph. Left: a simple graph allows at most one directed edge per ordered pair of nodes. Right: a multigraph allows parallel edges, so a pair of nodes can be connected by several distinct edges at once. Collapsing parallel edges into a single weighted edge would discard per-edge information. Preserving each edge explicitly retains it.

2.2.2. Message Passing

The *message passing* framework provides a unified view of GNN architectures. At each layer k, every node collects the representations of its neighbours, aggregates them into a single vector, and updates its own representation. Informally, each node listens to its neighbours, summarises what it hears, and updates itself. Formally,

$$\boldsymbol{h}_v^{(k)} = \text{UPDATE}\Big(\boldsymbol{h}_v^{(k-1)},\ \text{AGGREGATE}\big(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}\big)\Big). \tag{2.5}$$

Because $\mathcal{N}(v)$ has no natural ordering, AGGREGATE must be *permutation-invariant*, meaning it returns the same result regardless of the order in which neighbours are processed. Common choices are sum, mean, and element-wise maximum. Figure 2.2 shows one round of message passing for a single node.

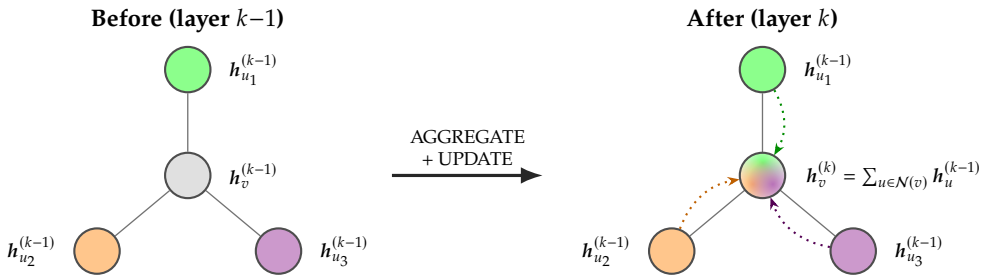

Figure 2.2: One round of message passing for node v. Left: the node and its three neighbours carry representations from layer $k-1$. Right: each neighbour sends its representation as a message (dotted arrow matching the neighbour's colour) to v. These messages are aggregated (e.g., by summation) and combined with v's own previous representation to produce its updated representation, shown as a blend of the neighbours' colours. Repeating this for K layers gives each node a representation summarising its K-hop neighbourhood.

The three components serve distinct roles. AGGREGATE collects neighbour representations into a single vector. UPDATE combines this aggregated message with the node's own previous representation. For graph-level tasks, a READOUT operation pools all node representations into a single graph-level vector. Since the paper focuses on node-level prediction, this operation is not developed further here.

After K layers, each node's representation $\boldsymbol{h}_v^{(K)}$ summarises the structure and features within its K-hop neighbourhood. GIN, introduced in Section 2.2.5, is a specific GNN architecture with a provably expressive aggregation function.

2.2.3. Inductive vs. Transductive Setting

GNNs can operate in two settings that differ in what information is available at training time. In the *transductive* setting, all nodes are present in the graph during training, even those whose labels are withheld as a test set. The model can therefore exploit the structural position of unlabelled nodes even before their labels are observed.

In the *inductive* setting, test nodes are completely unseen during training. The model must generalise from representations learned on the training subgraph to nodes it has never encountered. This reflects realistic financial monitoring, where new accounts appear after the model is deployed and must be classified without retraining. Figure 2.3 contrasts the two settings.

The paper operates in the inductive setting. Models are trained on a subset of transaction networks and evaluated on entirely separate, unseen networks, reflecting the realistic scenario where a deployed model must generalise to transaction patterns it has never encountered during training.

2.2.4. Neighbourhood Sampling

Full-graph message passing requires loading the entire neighbourhood of every node into memory at each training step. For a K-layer GNN, the receptive field of a single node expands exponentially with depth, making this approach infeasible on large graphs.

Neighbourhood sampling bounds this cost by fixing a maximum fan-out S: at each GNN layer, only a randomly drawn subset $\mathcal{N}_S(v) \subseteq \mathcal{N}(v)$ of at most S neighbours is used. A mini-batch is formed by first

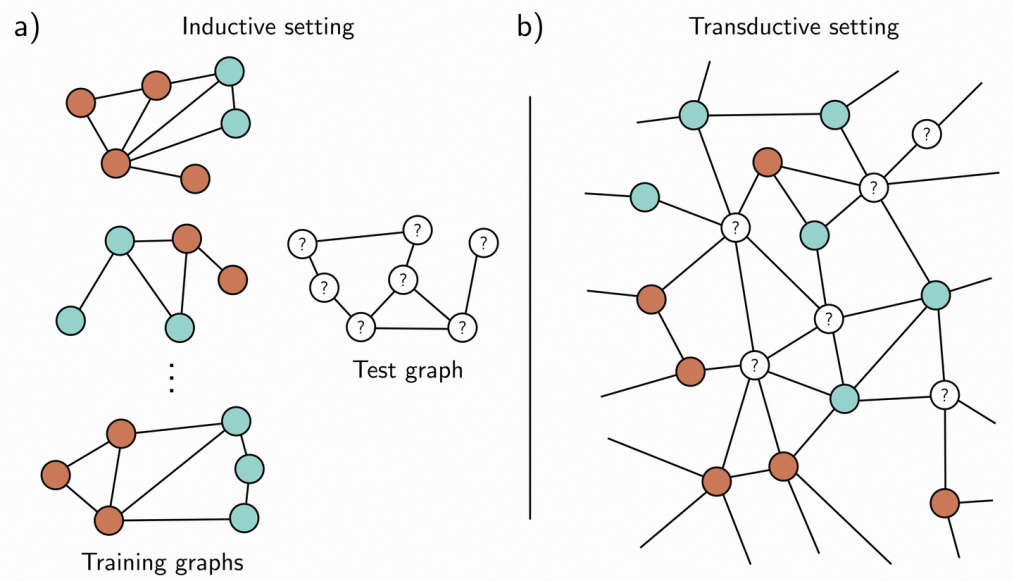

Figure 2.3: Inductive vs. transductive setting, from Prince [15]. In the inductive setting (a), the test graph is entirely absent during training. The model must generalise to completely unseen graph structures. In the transductive setting (b), all nodes appear in the same graph during training, including unlabelled test nodes, giving the model access to their structural position. The paper uses the inductive setting, which reflects the realistic scenario where the model is evaluated on entirely separate, unseen transaction networks.

sampling a set of target nodes, then tracing back through K rounds of neighbourhood sampling to collect the nodes needed to compute their representations. The GNN forward pass, loss computation, and parameter update all operate on this subgraph, keeping memory requirements independent of the global graph size [10]. Figure 2.4 illustrates the principle.

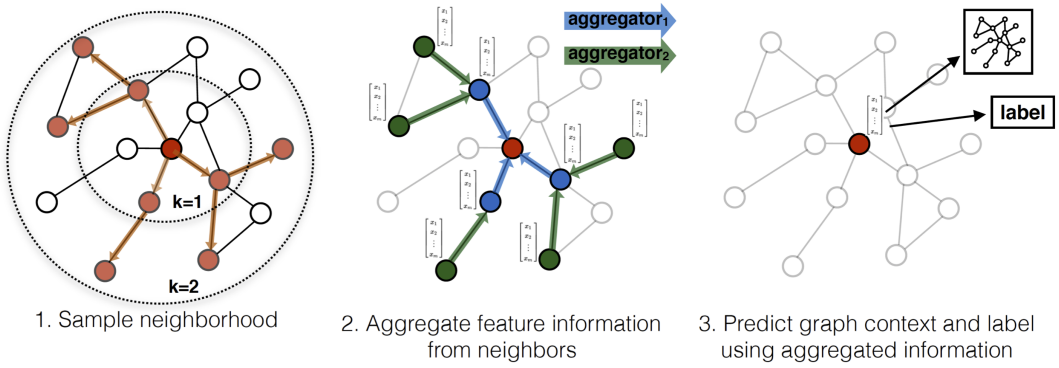

Figure 2.4: Sample-and-aggregate around a target node (red), as introduced by Hamilton et al. [10]. *Left:* at each hop the GNN draws at most S neighbours (the fan-out), building a fixed-size K-hop neighbourhood ($k = 1$, $k = 2$) rather than using the full graph. *Middle:* feature information is aggregated inward along the sampled edges, hop by hop, with a learned aggregator per hop. *Right:* the resulting representation predicts the target's label. This sampled subgraph replaces the full graph in the forward pass, loss, and parameter update, keeping memory requirements fixed regardless of graph size.

The paper applies this technique to multigraphs to enable training and adversarial evaluation on large financial networks.

2.2.5. Graph Isomorphism Network

Standard GNN architectures use mean or max aggregation, which can map structurally different neighbourhoods to identical representations. Consider two rooted subgraphs that differ only in the number of neighbours sharing the same feature value, say one centre with three such neighbours versus one with a single such neighbour. Mean aggregation returns the same average for both. Max aggregation returns the same maximum. Neither can distinguish them. Sum aggregation, however,

produces different totals for the two cases, separating them. Xu et al. [20] formalise this, showing that mean and max aggregators discard multiplicity and are strictly less expressive, whereas *sum* aggregation avoids the collapse and makes GIN provably as expressive as the Weisfeiler-Leman graph isomorphism test [18, 20].

The GIN update at layer k applies an MLP to a sum of two terms: the node's own previous representation up-weighted by $1 + \epsilon^{(k)}$, and the sum of its neighbours' previous representations.

$$\boldsymbol{h}_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \boldsymbol{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}_u^{(k-1)} \right). \tag{2.6}$$

Here $\mathrm{MLP}^{(k)}$ is a multilayer perceptron (Section 2.1.1) specific to layer k, $\epsilon^{(k)}$ is a learnable scalar, and $\boldsymbol{h}_v^{(0)}$ is the initial node feature vector, which is row v of $\boldsymbol{X}$.

For node-level prediction, the final representations $\boldsymbol{h}_v^{(K)}$ are passed through a classification head, a learned linear projection followed by a sigmoid activation, producing one real-valued prediction per label. Chapter 3 extends this architecture to multigraphs by replacing adjacency-matrix aggregation with per-edge message passing using an incidence-matrix reformulation that preserves multi-edge structure.

2.3. Adversarial Robustness on GNNs

Adversarial robustness studies what happens when an attacker deliberately modifies the input to fool a model, and how models can be trained to resist such manipulation. This section introduces the threat model that defines what the attacker may do, the gradient-based methods used to find effective perturbations, and adversarial training as a strategy for building more robust models.

2.3.1. Adversarial Attacks

This section characterises the attacker through the threat model that constrains their actions, and then describes the gradient-based optimisation used to find effective perturbations.

Threat Model

A threat model precisely specifies the rules under which an attacker operates. The concept of adversarial attacks originated in the image domain, where Goodfellow et al. [8] demonstrated that imperceptible perturbations to an image's pixels could cause a neural network to misclassify it with high confidence. Figure 2.5 illustrates this idea. A carefully crafted noise pattern, invisible to the human eye when added to an image, completely changes the model's prediction. The same principle applies to graph-structured data, where small, targeted modifications to a graph's edges can cause a GNN to misclassify nodes with high confidence.

The attacker modifies only the graph structure. In the simple-graph setting studied by prior work this means adding or deleting edges, while node features, node identities, and edge attributes remain fixed. The total number of modifications is bounded by a budget $\Delta \in \mathbb{N}$, fixed before the attack begins. In the financial multigraph setting of Chapter 3, unconstrained edge deletion is unrealistic, since a settled transaction cannot be erased from the banking record; the attack there instead *rewires* edges, redirecting a transaction to a different counterparty, which changes structure without removing past activity or introducing new edge attributes. The paper assumes the *white-box* setting, in which the attacker has full access to the model's parameters θ, its architecture, and the training data, making loss gradients with respect to the graph structure available. This is a conservative choice, since a white-box attacker is strictly stronger than a black-box one who can only observe model outputs, providing a meaningful robustness upper bound. The attacker's goal is *untargeted*, meaning it aims to degrade classification performance across all nodes rather than flipping the label of any specific one, subject to the budget Δ.

Edge Perturbation as an Optimisation Problem

The attack can be stated as an optimisation problem. Following [19], let $\boldsymbol{S} \in \{0,1\}^{n \times n}$ be a symmetric matrix encoding which edges are modified, where $\boldsymbol{S}_{ij} = 1$ means edge (i, j) is flipped (added if absent, removed if present), and $\boldsymbol{S}_{ij} = 0$ means it is left unchanged. Letting $\bar{\boldsymbol{A}} = \mathbf{1}\mathbf{1}^\top - \boldsymbol{I} - \boldsymbol{A}$ denote the

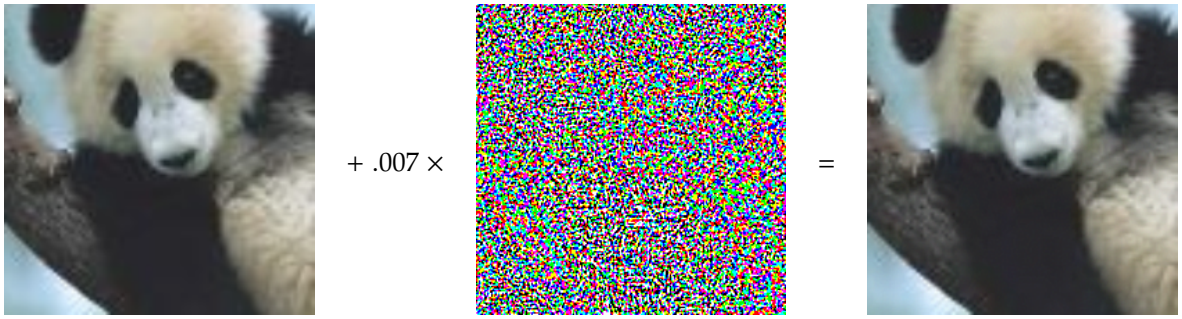

Figure 2.5: An adversarial example in the image domain, illustrating the concept introduced by Goodfellow et al. [8]. A panda image (left) is correctly classified with high confidence. Adding the sign of the loss gradient, scaled by ε, as an imperceptible noise pattern produces a perturbed image (right) that is visually indistinguishable from the original yet fools the model into predicting "gibbon" with 99.3% confidence. The same principle motivates structural attacks on GNNs, where edges play the role of pixels.

complement adjacency and $\boldsymbol{C} = \bar{\boldsymbol{A}} - \boldsymbol{A}$, entry $\boldsymbol{C}_{ij}$ equals $+1$ for non-edges and -1 for existing edges, so $\boldsymbol{A}' = \boldsymbol{A} + \boldsymbol{C} \circ \boldsymbol{S}$ adds the selected non-edges and removes the selected existing ones, where $\circ$ denotes the element-wise product. The search space is combinatorial. Within a budget of Δ edge changes, the number of candidate perturbation sets grows exponentially, making exhaustive search infeasible.

The standard solution is a *continuous relaxation*. Instead of searching over binary $\boldsymbol{S} \in \{0,1\}^{n \times n}$, we relax to $\boldsymbol{S} \in [0,1]^{n \times n}$, allowing gradient-based optimisation. At the end, the learned real-valued entries of $\boldsymbol{S}$ are used as sampling probabilities to draw a discrete binary perturbation. The relaxed attack objective is

$$\min_{\boldsymbol{S} \in \mathcal{P}} \mathcal{L}_{\text{attack}}\big(f_\theta(\boldsymbol{A} + \boldsymbol{C} \circ \boldsymbol{S},\, \boldsymbol{X}),\, \boldsymbol{y}\big), \tag{2.7}$$

where f_θ is the GNN, $\boldsymbol{y}$ are the ground-truth labels, $\mathcal{L}_{\text{attack}}$ is an attack loss that is low when the GNN misclassifies many nodes, and $\mathcal{P} = \big\{\boldsymbol{S} \in [0,1]^{n \times n} : \boldsymbol{S} = \boldsymbol{S}^\top,\, \sum_{i<j} \boldsymbol{S}_{ij} \le \Delta\big\}$. Figure 2.6 illustrates the effect of a structural perturbation on a graph. PGD [19, 14] and PR-BCD [7] are two solvers for this objective, differing in how they traverse the space of candidate edge perturbations.

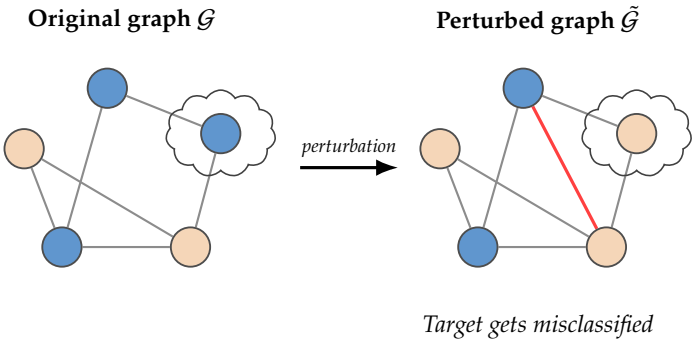

Figure 2.6: Effect of a structural perturbation on node classification. Left: the clean graph $\mathcal{G}$ with correct predictions for all nodes (class A in blue and class B in peach); the cloud marks the attack target. Right: adding one edge (red) by solving (2.7) shifts the target's neighbourhood, and the GNN now predicts it as class A instead of class B. The figure shows a single targeted node for visual clarity; the attacks studied in this work are untargeted, degrading predictions across many nodes at once rather than flipping any one node in particular. Figure adapted from [21].

Projected Gradient Descent Attack

Projected gradient descent (PGD) [19, 14] solves the attack objective (2.7) by iteratively moving the perturbation matrix $\boldsymbol{S}$ along the gradient descent direction and then projecting the result back onto the

feasible set $\mathcal{P}$. Starting from $\boldsymbol{S}^{(0)} = \mathbf{0}$, each step reads

$$\boldsymbol{S}^{(t+1)} = \mathrm{proj}_{\mathcal{P}}\Big(\boldsymbol{S}^{(t)} - \alpha \, \nabla_{\boldsymbol{S}^{(t)}} \mathcal{L}_{\text{attack}}\big(f_\theta(\boldsymbol{A} + \boldsymbol{C} \circ \boldsymbol{S}^{(t)}, \, \boldsymbol{X}), \, \boldsymbol{y}\big)\Big), \tag{2.8}$$

where $\alpha > 0$ is the step size and $\mathrm{proj}_{\mathcal{P}}$ is the Euclidean projection onto $\mathcal{P}$,

$$\mathrm{proj}_{\mathcal{P}}(\boldsymbol{S}) = \arg\min_{\boldsymbol{S}' \in \mathcal{P}} \; \tfrac{1}{2}\|\boldsymbol{S}' - \boldsymbol{S}\|_2^2, \tag{2.9}$$

computed efficiently via the capped-simplex projection algorithm of Ang et al. [2].

Figure 2.7 illustrates PGD as gradient descent in the space of perturbations $\boldsymbol{S}$. Each step moves toward lower attack loss and projects back into the feasible region until the solution converges.

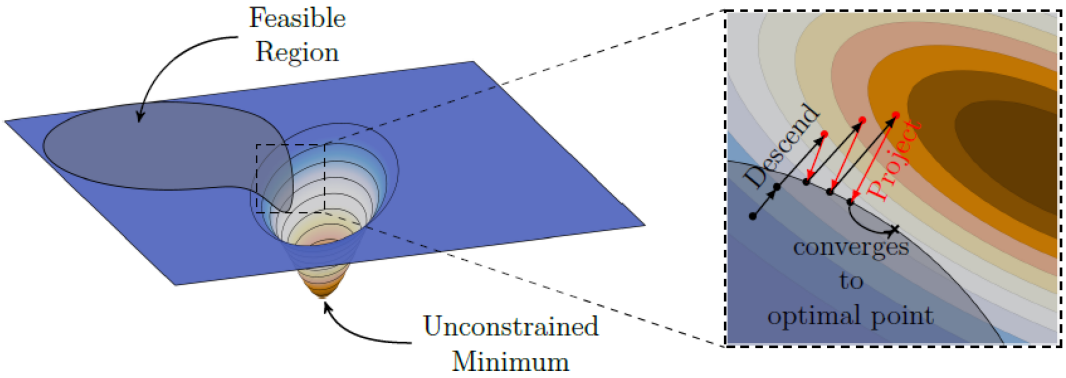

Figure 2.7: PGD attack as gradient descent in the perturbation space $\boldsymbol{S}$, illustrated in Friedrich et al. [6]. The attack loss $\mathcal{L}_{\text{attack}}$ has an unconstrained minimum that falls outside the feasible region (left). Each iteration descends along the gradient and projects back onto the feasible region boundary (right). The procedure converges to the optimal point within the feasible region, after which the continuous entries of $\boldsymbol{S}^*$ are used as sampling probabilities to draw the discrete binary perturbation.

PGD computes gradients with respect to all $O(n^2)$ entries of $\boldsymbol{A}$ simultaneously, which becomes memory-intensive for large graphs.

Projected Randomised Block Coordinate Descent Attack

Projected randomised block coordinate descent (PR-BCD) [7] is a scalable alternative to PGD. Instead of updating all $O(n^2)$ candidate edge entries simultaneously, PR-BCD randomly selects a small *block* of candidate edges at each step and restricts gradient computation to that block alone. The block is re-sampled at every iteration, so that different parts of the edge space are explored over the course of the attack.

At each step, the gradient update and projection are applied only to the entries of $\boldsymbol{S}$ belonging to the sampled block, with the total budget constrained to at most Δ edge flips. At the end, the continuous entries of $\boldsymbol{S}^*$ are used as sampling probabilities to draw a discrete binary perturbation.

PR-BCD scales to large graphs where full-graph gradient computation is infeasible, at the cost of exploring only a random subset of candidate edges per step, which may yield a locally rather than globally optimal perturbation.

2.3.2. Adversarial Training

Adversarial training hardens the model against structural attacks by incorporating worst-case perturbations into the training objective itself. Figure 2.8 illustrates the core idea.

Min-Max Formulation

Adversarial training finds model parameters that remain effective even when an adversary selects the most damaging perturbation. At each step, the attacker maximises the classification loss $\mathcal{L}_{\text{cls}}$ over the graph structure to find the perturbation $\boldsymbol{S}^*$ that most degrades the current model. The model is then updated so that it performs well on this worst-case perturbed graph. The objective is [14, 9]:

$$\arg\min_{\theta} \; \max_{\boldsymbol{S} \in \mathcal{P}} \; \mathcal{L}_{\text{cls}}\big(f_\theta(\boldsymbol{A} + \boldsymbol{C} \circ \boldsymbol{S}, \, \boldsymbol{X}), \, \boldsymbol{y}\big), \tag{2.10}$$

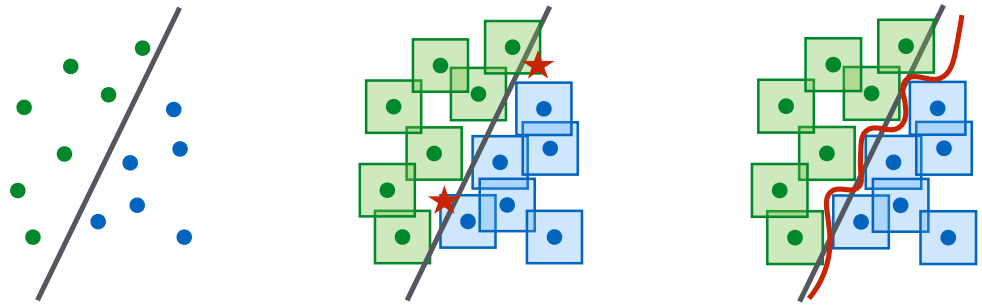

Figure 2.8: A conceptual illustration of standard versus adversarially robust decision boundaries, from Madry et al. [14]. Left: Data points from two classes that can be easily separated by a simple linear decision boundary (gray). Middle: The simple decision boundary does not separate the perturbation regions (squares) around the data points. The red stars mark adversarial examples within these regions that are misclassified. Right: A robust decision boundary (red) that correctly separates all perturbation regions. Adversarial training learns such a boundary by exposing the model to worst-case perturbations at every training iteration.

where $\mathcal{P}$ encodes both the budget constraint $\sum_{i<j} \boldsymbol{S}_{ij} \leq \Delta$ and the relaxed domain $[0,1]^{n\times n}$. The inner maximisation over $\boldsymbol{S}$ finds the most damaging perturbation for the current model. The outer arg min over θ then finds parameters that perform well even under this worst-case graph. The inner maximisation is solved approximately using PR-BCD, producing a perturbation $\boldsymbol{S}^*$ used to update θ.

Adversarial Training on Graphs

In practice, the objective (2.10) is solved by alternating between the inner attack and the outer model update. Figure 2.9 shows the complete training loop.

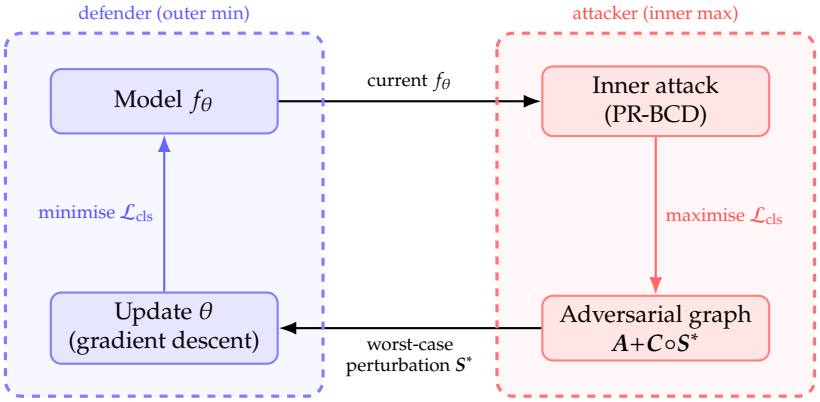

Figure 2.9: The adversarial training loop. At each iteration, the inner attack (PR-BCD) maximises the classification loss $\mathcal{L}_{\text{cls}}$ to find the worst-case perturbation $\boldsymbol{S}^*$ for the current model f_θ (red). The model parameters θ are then updated by minimising $\mathcal{L}_{\text{cls}}$ on this perturbed graph (blue). Repeating this loop produces a model hardened against the class of structural perturbations the inner solver can produce.

The alternating procedure runs as follows:

1. Run T steps of gradient ascent on $\mathcal{L}_{\text{cls}}$ using PR-BCD on the current model f_θ to find an approximate worst-case perturbation $\boldsymbol{S}^*$.
2. Sample a discrete binary perturbation from $\boldsymbol{S}^*$ using its entries as probabilities.
3. Update the model parameters via gradient descent on $\mathcal{L}_{\text{cls}}(f_\theta(\boldsymbol{A} + \boldsymbol{C} \circ \boldsymbol{S}^*, \boldsymbol{X}), \boldsymbol{y})$.
4. Repeat until convergence.

PR-BCD restricts each gradient step to a randomly selected block of candidate edges, enabling the inner maximisation to scale to large graphs at the cost of some optimality.

This training procedure, applied with PR-BCD as the inner solver on financial transaction multigraphs, is the defence evaluated in Chapter 3.

2.4. Money Laundering Patterns

Money laundering leaves recurring structural traces in a transaction network. To disguise the origin of illicit funds, launderers route money through chains of accounts whose connection structure follows a small number of recurring shapes, largely independent of the amounts moved or the particular accounts involved [5, 4]. Figure 2.10 shows the patterns that the detection task in Chapter 3 targets. Each pattern is a small directed subgraph, and the task is to decide, for a given account, whether the account occupies the marked position inside such a subgraph. Detecting these shapes in the transaction graph is the concrete prediction task the multigraph GNN of Chapter 3 learns to solve.

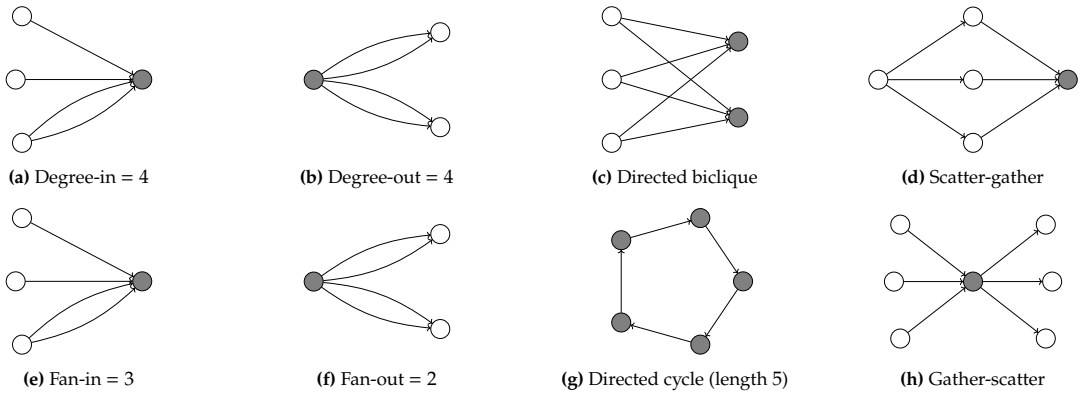

Figure 2.10: Money laundering patterns, following the synthetic pattern taxonomy of Egressy et al. [5]. Each subfigure is a small directed subgraph in which an arrow denotes a transaction from one account to another, and the gray account is the node a detector must flag as occupying the marked position. Degree-in and degree-out count incoming and outgoing transactions; fan-in and fan-out count distinct counterparties rather than transactions; scatter-gather and gather-scatter route funds through a layer of intermediaries; the directed cycle returns funds to their origin; and the biclique connects two groups of accounts all-to-all. The degree, fan, and cycle sizes shown are illustrative; Chapter 3 fixes a threshold per pattern. The same structural shapes recur as the targets that anti-money laundering detectors are built to find.

Each pattern abstracts a documented laundering typology. Fan-in and fan-out capture *smurfing*, where a launderer splits a large sum into many small transfers across distinct counterparties to keep each transaction below the threshold that triggers a regulatory report [11]. Scatter-gather and gather-scatter capture *layering*, where funds are dispersed across a set of intermediary accounts and recombined at a single destination, severing the direct link between source and beneficiary; Egressy et al. [5] illustrate scatter-gather with a hotel that mixes illicit cash into legitimate revenue before forwarding it to a criminal owner. The directed cycle captures *round-tripping*, where money passes through a chain of accounts and returns to its origin, restoring control over the funds while manufacturing an appearance of unrelated trade. Degree-in and degree-out capture raw transaction volume in one direction, and the biclique captures dense all-to-all transfer between two groups of accounts. The recurring shapes give an automated detector a structural signature to search for, even when the launderer varies amounts and accounts to evade fixed rules.

The patterns are deliberate simplifications of real laundering schemes rather than literal labels of criminal activity. Egressy et al. [5] instantiate the patterns as a synthetic benchmark, generating transaction multigraphs in which every account carries a known binary label for each pattern, so that a detector can be trained and measured against an exact ground truth. Blanuša et al. [4] mine the same shapes, including cycles and scatter-gather patterns, as real-time engineered features in a deployed financial-crime system, evidence that the patterns carry detection signal beyond the synthetic setting. Chapter 3 evaluates on the synthetic benchmark of Egressy et al., where each account carries one label for each of eleven patterns: in-degree, out-degree, fan-in, fan-out, directed cycles of length two through six, scatter-gather, and biclique. A label is positive when the account occupies the marked position in the corresponding subgraph, and the multigraph GNN predicts all eleven labels per account at once.

Detecting the patterns demands an expressive graph neural network. Separating a fan from a raw degree requires counting distinct counterparties rather than transactions, and detecting a directed cycle requires tracing a path that returns to its start; the mean and max aggregators of Section 2.2.5 discard the multiplicities that both distinctions depend on, whereas the sum aggregation of GIN preserves them. Counting parallel transactions between the same pair of accounts further requires the per-edge

multigraph message passing that Chapter 3 develops, since collapsing parallel edges into a single adjacency entry erases the counts the patterns are defined by. The patterns of Figure 2.10 therefore motivate the expressive multigraph GNN that Chapter 3 both attacks and defends.

3

Scientific Paper

This chapter presents the scientific paper that represents the core contribution of this thesis. The evaluation uses the synthetic pattern detection benchmark. Extending the evaluation to attribute-rich datasets such as AMLworld [1], lies outside the scope of this thesis and forms part of an extended version of the paper that we aim to submit to the AAAI Conference on Artificial Intelligence.

Adversarial Robustness of Multigraph Neural Networks

Dennis Heijmans
Delft University of Technology
Delft, The Netherlands
dheijmans@tudelft.nl

H. Çağrı Bilgi
Delft University of Technology
Delft, The Netherlands
cbilgi@tudelft.nl

Rui Wang
Delft University of Technology
Delft, The Netherlands
rwang14@tudelft.nl

Kubilay Atasu
Delft University of Technology
Delft, The Netherlands
katasu@tudelft.nl

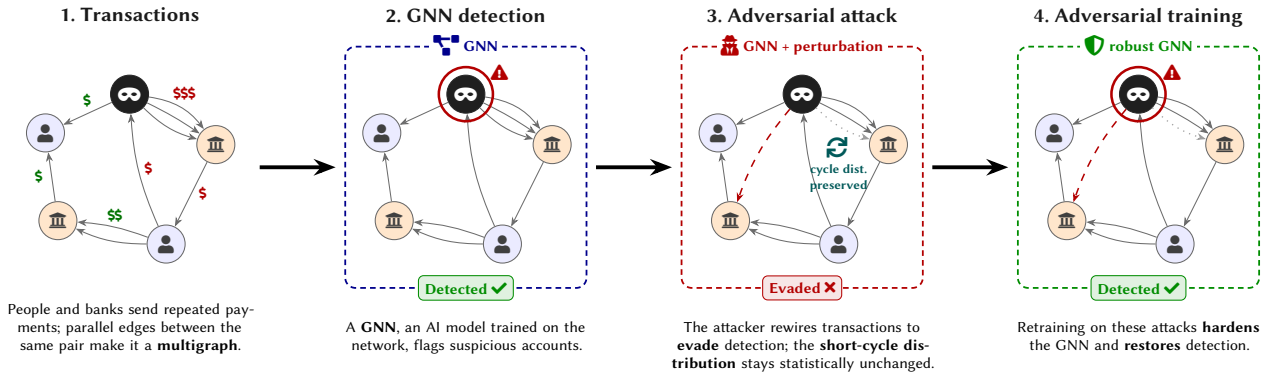

Figure 1: Overview of our attack-and-defense framework for multigraph GNNs in anti-money laundering. Real financial transactions form a multigraph; a GNN flags suspicious accounts; an adversary rewires transactions to evade detection while a cycle unnoticeability term keeps the short-cycle distribution statistically unchanged; and adversarial training restores robustness against these structural attacks.

Abstract

Anti-money laundering systems built on financial transaction networks must be robust against adversaries who can manipulate transaction structure to evade detection. Graph neural networks (GNNs) for this task increasingly operate on multigraphs, where multiple parallel transactions can exist between the same account pair, but existing adversarial robustness methods operate on the adjacency matrix, which records at most one edge per node pair and cannot represent these parallel transactions. Despite the recent development of multigraph GNNs, structural adversarial attacks on multigraph representations remain unexplored. We propose a scalable framework for structural attacks and adversarial training on multigraph GNNs. The framework uses an incidence-matrix representation that makes node embeddings differentiable with respect to graph structure, extends projected gradient descent (PGD) and projected randomized block coordinate descent (PR-BCD) attacks to the multigraph setting, and introduces unnoticeability loss terms that constrain perturbations to preserve the macro-level statistical fingerprint of the original graph, keeping them undetectable by statistical monitoring systems. We evaluate the framework on a synthetic pattern detection benchmark in a fully inductive setting. Learned attacks reduce macro PR-AUC from 99% to 80% at a 10% rewiring budget, substantially outperforming non-learnable perturbations, and adversarial training recovers robustness to 88%, demonstrating that multigraph GNNs are vulnerable to structural manipulation and that this vulnerability can be addressed through adversarial training.

1 Introduction

Money laundering channels the proceeds of crime through the financial system, and the illicit transfers hide among the billions of legitimate payments that move between accounts every day. Estimated to account for 2% to 5% of global GDP annually, laundering proceeds sustain organized crime, drug trafficking, corruption, and terrorist financing [15]. Detecting them increasingly relies on automated analysis of the transaction networks those payments form. Graph neural networks (GNNs) have emerged as the leading tool for the task, reaching high detection accuracy on structural pattern analysis and transaction classification [2–4].

Financial transaction networks are naturally multigraphs. The same pair of accounts may exchange many payments, loans, or transfers over time, producing multiple parallel edges between one node pair. Each edge is a distinct transaction carrying its own amount, timestamp, and direction. A multigraph differs from a simple graph, which allows at most one edge per node pair, and from a hypergraph, where a single edge connects more than two nodes; only the multigraph captures repeated directed transactions between the same two accounts. The per-edge amounts, timestamps,

and transfer frequencies are essential for detecting suspicious patterns, which become invisible once repeated transactions are collapsed into a single aggregate connection. Multigraph structure also arises beyond finance, in social interaction and transportation networks where entities interact repeatedly [2].

Recent GNNs for financial crime detection operate directly on the multigraph through per-edge message passing, where each transaction contributes its own message rather than being merged with other transactions between the same accounts [2–4]. Per-edge message passing contrasts with approaches that first aggregate parallel edges into a single weighted connection per node pair, reducing the multigraph to a weighted graph representable by a standard adjacency matrix [13]. Per-edge GNNs distinguish transaction patterns that adjacency-matrix methods cannot, making them strictly more expressive. Their computation has no adjacency-matrix representation, however, so the standard toolkit for analyzing model robustness does not apply to them.

Financial criminals do not passively accept detection: money laundering schemes are deliberately structured to evade automated monitoring [10]. Breaking large transfers into many smaller ones, routing funds through intermediary accounts, and timing transactions to fall below reporting thresholds are documented evasion tactics that amount to reshaping the transaction graph. The threat raises a concrete question: can an adversary manipulate the structure of a financial transaction network, by adding or removing individual transactions, to cause a multigraph GNN to miss illicit activity? A detector that can be systematically evaded by such structural manipulation provides only illusory protection, validating laundering schemes precisely when adversarial pressure is highest. Structural attacks on GNNs formalise the threat. On simple graphs, small, targeted structural perturbations can substantially degrade the accuracy of a GNN [5, 7, 11, 17].

Existing work on structural attacks and adversarial defenses for GNNs is formulated entirely in terms of the adjacency matrix and therefore applies only to simple graphs. Adversarial robustness on hypergraphs is a separate problem, with a different graph structure and a different threat model [14]. Per-edge multigraph GNNs fit neither formulation, so existing attack and defense algorithms are incompatible with them at a foundational level. Their adversarial robustness has remained unstudied, despite the growing adoption of multigraph GNNs for financial crime detection.

Our work. We close this gap by proposing the first framework for structural attacks and adversarial training on per-edge multigraph GNNs, built around three contributions.

Structural attacks on multigraph GNNs. We introduce the first gradient-based structural attack framework that is natively compatible with per-edge multigraph GNNs. Existing attack methods differentiate through adjacency-matrix entries, which cannot represent distinct parallel edges and are therefore incompatible with per-edge message passing. We resolve this incompatibility by reformulating multigraph GNN computation using the incidence matrix, which preserves full edge multiplicity and makes node embeddings differentiable with respect to the multigraph structure through a continuously relaxed incidence matrix.

Cycle unnoticeability. We introduce unnoticeability loss terms that constrain perturbations to leave the statistical distribution of short cycles of length two and three invariant before and after attack. Counting cycles of a given length from the graph structure is tractable for small lengths via closed-form expressions [9, 16]. Cycle-count statistics form part of the macro-level fingerprint of a financial transaction network, and a monitoring system that inspects these statistics independent of the GNN would register a perturbation that alters them [11]. Maintaining cycle unnoticeability keeps the perturbed graph within the statistical distribution of the original, preventing trivial detection by such a monitor and ensuring that measured degradation reflects model confusion rather than a detectable structural shift. The framework naturally extends to additional unnoticeability terms for other graph statistics.

Scalable attacks and adversarial defenses. We scale the attack framework to large financial networks by adapting projected randomized block coordinate descent to the incidence-matrix setting, reducing memory requirements so that they no longer grow with network size. We further demonstrate adversarial training on multigraph GNNs, showing that adversarial training hardens models against the proposed structural attacks within the same framework, enabling both attack evaluation and adversarial defense.

We evaluate the framework on the IBM synthetic pattern detection benchmark in a fully inductive setting. Learned attacks reduce macro PR-AUC from 99% to 80% at a 10% rewiring budget, substantially outperforming non-learned perturbations, and adversarial training recovers robustness to 88%.

2 Background and Related Work

Graph neural networks (GNNs) have become a leading tool for fraud detection in financial transaction networks, where accounts are nodes and transactions are edges [2–4]. Among GNN architectures, Graph Isomorphism Networks (GIN) [18] are among the most expressive, and form the basis of the multigraph GNN of [4]. Their layer update is

$$\boldsymbol{H}^{(k+1)} = \mathrm{MLP}^{(k)}\Big((1+\epsilon^{(k)})\boldsymbol{H}^{(k)} + \boldsymbol{A}\boldsymbol{H}^{(k)}\Big), \tag{1}$$

where $\boldsymbol{H}^{(k)} \in \mathbb{R}^{n\times d}$ is the node feature matrix and $\boldsymbol{A} \in \{0,1\}^{n\times n}$ is the binary adjacency matrix. The adjacency matrix collapses all parallel edges into a single entry per node pair, discarding transaction-level information such as individual amounts, timestamps, and frequencies that are essential for anti-money laundering detection.

Egressy et al. [4] extend GIN to multigraphs by replacing adjacency-matrix aggregation with per-edge message passing. Each transaction edge contributes an individual message that combines its source node's features with its own edge features, and target nodes aggregate over all incoming edges without collapsing parallel connections. Because this aggregation iterates directly over the edge multiset $\mathcal{E}$, no adjacency-matrix representation of the computation exists. Gradient-based structural attack methods, which differentiate through the entries of $\boldsymbol{A}$, are therefore fundamentally incompatible with this per-edge formulation.

Early work on adversarial robustness was driven by discoveries in the image domain, where small but carefully constructed perturbations were shown to mislead neural networks [6, 12]. This motivated a broader investigation into adversarial vulnerabilities

across modalities, including graphs. Within the GNN domain, research has examined how structural manipulations can compromise model reliability [5, 7, 11, 17].

Gradient-based structural attacks were shown to substantially reduce GNN accuracy [17], with subsequent work improving scalability [5] and introducing adversarial training strategies that harden GNNs against them [7]. The field has since recognized that robustness evaluation requires perturbations that are realistic and unnoticeable rather than arbitrary graph modifications [11]. Robustness research also extends beyond simple graphs to multi-relational structures, with adversarial attacks developed for knowledge-graph embeddings [20] and heterogeneous graphs [19]. These approaches represent structure as one adjacency matrix per relation type, so they too cannot perturb the parallel edges of a single relation that the per-edge multigraph setting requires.

Multigraph GNNs fall into two categories. The first aggregates parallel edges into a single weighted connection per node pair, reducing the multigraph to a weighted graph with an adjacency-matrix representation [13]. Such methods are compatible with standard adjacency-matrix-based adversarial attack and defense algorithms. The second category preserves full edge multiplicity through per-edge message passing, using an edge-indexed representation that replaces the adjacency matrix altogether [2–4]. Existing adversarial training and attack methods operate on adjacency matrices and are therefore fundamentally incompatible with this edge-indexed representation, making it impossible to evaluate or adversarially train per-edge multigraph GNNs using existing approaches. Our work fills this gap by proposing a framework for structural attacks and adversarial training that operates directly on multigraph structure, without collapsing edge multiplicity.

3 Problem Statement

Consider a directed, unweighted multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is a set of n nodes and $\mathcal{E} = \{\{e_1, \dots, e_m\}\}$ is a multiset of m directed edges. Each edge $e_j = (u_j, w_j)$ consists of a source node $u_j \in \mathcal{V}$ and a target node $w_j \in \mathcal{V}$. Let f_θ be a multigraph GNN and let $\boldsymbol{y}$ denote the ground-truth labels. We consider a white-box evasion setting: the attacker has full access to f_θ and θ is fixed during the attack.

A structural perturbation rewires a subset of edges by replacing their target nodes while keeping source nodes unchanged. Rewiring keeps the number of edges fixed, so the perturbation is a fixed-dimensional variable amenable to first-order optimisation over the incidence matrix; it also matches the anti-money-laundering setting, where an adversary redirects future transactions rather than erasing settled ones. Given a rewiring budget $\Delta \in \mathbb{N}$, the adversarial attack problem is

$$\max_{\widetilde{\mathcal{G}} \in \mathcal{B}(\mathcal{G}, \Delta)} \mathcal{L}_{0/1}\big(f_\theta(\widetilde{\mathcal{G}}), \boldsymbol{y}\big), \tag{2}$$

where $\mathcal{B}(\mathcal{G}, \Delta)$ denotes the set of all multigraphs obtainable from $\mathcal{G}$ by rewiring at most Δ edge targets. The adversarial training problem, which seeks model parameters robust to worst-case perturbations within the same budget, is

$$\arg\min_{\theta} \max_{\widetilde{\mathcal{G}} \in \mathcal{B}(\mathcal{G}, \Delta)} \mathcal{L}_{0/1}\big(f_\theta(\widetilde{\mathcal{G}}), \boldsymbol{y}\big). \tag{3}$$

4 Structural Attacks on Multigraph GNNs

4.1 Incidence-Based GNN Reformulation

We reformulate the per-edge message passing of the multigraph GIN [4] using the oriented incidence matrix $\boldsymbol{B} \in \mathbb{R}^{n \times m}$, whose rows correspond to nodes and whose columns correspond to edges. The entries of $\boldsymbol{B}$ are defined as

$$\boldsymbol{B}_{ij} = \begin{cases} -1 & \text{if edge } e_j \text{ leaves node } v_i, \\ +1 & \text{if edge } e_j \text{ enters node } v_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

With one column assigned to each edge, parallel edges are naturally distinguished and the computation becomes expressible as matrix products rather than an iteration over the edge multiset. The original edge-list representation does not support differentiation with respect to the graph structure, which gradient-based structural attacks require. To enable such attacks, we decompose $\boldsymbol{B}$ into its positive and negative parts: $\boldsymbol{B}_{\text{dst}} \in \{0,1\}^{n \times m}$, where $(\boldsymbol{B}_{\text{dst}})_{ij} = 1$ if e_j enters v_i, and $\boldsymbol{B}_{\text{src}} \in \{0,1\}^{n \times m}$, where $(\boldsymbol{B}_{\text{src}})_{ij} = 1$ if e_j leaves v_i (both zero otherwise), so $\boldsymbol{B} = \boldsymbol{B}_{\text{dst}} - \boldsymbol{B}_{\text{src}}$.

At each layer k, node features propagate forward along outgoing edges, combine with edge features before a nonlinearity, and aggregate at the destination nodes. The same process runs in reverse along incoming edges. Both messages are summed with a skip connection and passed through an MLP to produce the updated node features. The message passing update at layer k is defined as

$$\begin{aligned}
\boldsymbol{M}_{\text{fwd}}^{(k)} &= \phi\Big(\boldsymbol{B}_{\text{src}}^{\top} \boldsymbol{H}^{(k)} + \boldsymbol{Z}^{(k)}\Big), \\
\boldsymbol{H}_{\text{fwd}}^{(k)} &= \text{MLP}_{\text{fwd}}^{(k)}\Big(\big(1 + \epsilon_{\text{fwd}}^{(k)}\big)\boldsymbol{H}^{(k)} + \boldsymbol{B}_{\text{dst}} \boldsymbol{M}_{\text{fwd}}^{(k)}\Big), \\
\boldsymbol{M}_{\text{rev}}^{(k)} &= \phi\Big(\boldsymbol{B}_{\text{dst}}^{\top} \boldsymbol{H}^{(k)} + \boldsymbol{Z}^{(k)}\Big), \\
\boldsymbol{H}_{\text{rev}}^{(k)} &= \text{MLP}_{\text{rev}}^{(k)}\Big(\big(1 + \epsilon_{\text{rev}}^{(k)}\big)\boldsymbol{H}^{(k)} + \boldsymbol{B}_{\text{src}} \boldsymbol{M}_{\text{rev}}^{(k)}\Big), \\
\boldsymbol{H}^{(k+1)} &= \tfrac{1}{2}\left(\boldsymbol{H}^{(k)} + \phi\left(\text{BN}\left(\frac{\boldsymbol{H}_{\text{fwd}}^{(k)} + \boldsymbol{H}_{\text{rev}}^{(k)}}{2}\right)\right)\right), \\
\boldsymbol{Z}^{(k+1)} &= \boldsymbol{Z}^{(k)} + \tfrac{1}{2}\,\text{MLP}_{\text{edge}}^{(k)}\big[\boldsymbol{B}_{\text{src}}^{\top} \boldsymbol{H}^{(k+1)},\, \boldsymbol{B}_{\text{dst}}^{\top} \boldsymbol{H}^{(k+1)},\, \boldsymbol{Z}^{(k)}\big],
\end{aligned} \tag{5}$$

where $\boldsymbol{H}^{(k)} \in \mathbb{R}^{n \times d}$ are node features at layer k, $\boldsymbol{Z}^{(k)} \in \mathbb{R}^{m \times d}$ are edge features, ϕ is ReLU, BN is batch normalisation, $\text{MLP}_{\text{fwd}}^{(k)}$ and $\text{MLP}_{\text{rev}}^{(k)}$ are direction-specific two-layer perceptrons, $\text{MLP}_{\text{edge}}^{(k)}$ updates edge features from the concatenation of neighbouring node embeddings and the current edge embedding, and $\epsilon_{\text{fwd}}^{(k)}$, $\epsilon_{\text{rev}}^{(k)}$ are learnable scalars. Expressing the update in terms of $\boldsymbol{B}_{\text{src}}$ and $\boldsymbol{B}_{\text{dst}}$ makes node embeddings differentiable with respect to $\boldsymbol{B}$, so any loss on the node embeddings propagates gradients directly to the graph structure.

4.2 Structural Perturbations

To introduce perturbations to the graph structure, we employ a continuous relaxation of the discrete attack problem, inspired by [17]. Let $\boldsymbol{p} \in [0,1]^m$ denote a vector representing the probability of rewiring each edge, and $\boldsymbol{R} \in [0,1]^{m \times n}$ denote a matrix assigning scores to all nodes as potential new targets for each edge. We set $\boldsymbol{R}_{ji} = -\infty$ if v_i is either the source or target node of edge e_j, ensuring that an edge cannot be rewired to its original endpoints.

Applying a row-wise softmax to $\boldsymbol{R}$ then yields a probability distribution $\boldsymbol{Q}_j^{(t)}$ over all candidate target nodes for each edge e_j. We can then combine the probability vector $\boldsymbol{p}$ with the probability matrix $\boldsymbol{Q}$ to define a relaxed incidence matrix $\hat{\boldsymbol{B}}$, which encodes the expected structure of the perturbed graph. Specifically, for each edge e_j, if it is not rewired (with probability $1 - \boldsymbol{p}_j$), it remains connected to its original target node w_j. If the edge is rewired (with probability $\boldsymbol{p}_j$), it connects to a new target node v_i according to the probability $\boldsymbol{Q}_{ji}$. Formally, this relationship is given by

$$\hat{\boldsymbol{B}}_{ij}^{(t)} = \begin{cases} -1 & \text{if } v_i = u_j, \\ 1 - \boldsymbol{p}_j^{(t)} & \text{if } v_i = w_j, \\ \boldsymbol{p}_j^{(t)} \boldsymbol{Q}_{ji}^{(t)} & \text{otherwise.} \end{cases} \tag{6}$$

We optimize $\boldsymbol{p}$ and $\boldsymbol{R}$ jointly by gradient descent on a differentiable surrogate loss $\mathcal{L}'$ (Section 5), following [5]. After each update, we project the probability vector $\boldsymbol{p}$ onto the feasible set, ensuring that the expected total number of rewired edges remains within the allowed budget $\sum \boldsymbol{p} \leq \Delta$ and that $\boldsymbol{p} \in [0,1]^m$. We use the function Π_C by [1] for efficiently projecting $\boldsymbol{p}$ onto the capped simplex defined by these constraints. Formally, this projection step is given by

$$\begin{aligned} \boldsymbol{p}^{(t+1)} &\leftarrow \Pi_C(\boldsymbol{p}^{(t+1)}) \text{ where} \\ C &= \left\{ \boldsymbol{x} \in [0,1]^m \;\middle|\; \textstyle\sum_{j=1}^m \boldsymbol{x}_j \leq \Delta \right\}. \end{aligned} \tag{7}$$

Finally, we generate K discrete incidence matrices $\widetilde{\boldsymbol{B}}$ from the optimized probabilities $\boldsymbol{p}$ and $\boldsymbol{Q}$ to assess the attack's effectiveness. Here u_j and w_j denote the source and original target node of edge e_j. Each matrix is constructed by first sampling a binary vector $E^{(k)}$ that identifies which edges are rewired, while respecting the budget constraint, and then assigning new target nodes for the rewired edges according to $\boldsymbol{Q}$. We select the incidence matrix that yields the lowest model accuracy as the final adversarial instance. This sampling strategy is similar to that used by [17] for sampling adjacency matrices. Formally, the sampling procedure is given by Algorithm 1.

Algorithm 1 Adversarial incidence matrix selection using randomized perturbation sampling

Input: rewiring probabilities $\boldsymbol{p}$, target distributions $\boldsymbol{Q}$
Parameters: rewiring budget Δ, number of samples K
for $k = 1, 2, \ldots, K$ **do**
  Sample a binary mask indicating which edges to rewire:

$$\boldsymbol{E}^{(k)} \sim \text{Bernoulli}(\boldsymbol{p}) \quad \text{s.t.} \quad \sum_{j=1}^m E_j^{(k)} \leq \Delta \tag{8}$$

  Sample new target nodes for the selected edges:

$$\tilde{w}_j^{(k)} = \begin{cases} w_j & \text{if } E_j^{(k)} = 0 \\ v_i & \text{with probability } \boldsymbol{Q}_{ji} \end{cases}, \forall j \tag{9}$$

  Construct the perturbed incidence matrix:

$$\widetilde{\boldsymbol{B}}_{ij}^{(k)} = \begin{cases} -1 & \text{if } v_i = u_j \\ +1 & \text{if } v_i = \tilde{w}_j^{(k)} \\ 0 & \text{otherwise} \end{cases}, \forall i, j \tag{10}$$

end for
Output: Perturbed incidence matrix $\widetilde{\boldsymbol{B}}^{(k^*)}$ where

$$k^* = \arg\max_k \; \mathcal{L}_{0/1}\big(f_\theta(\widetilde{\boldsymbol{B}}^{(k)}, \boldsymbol{X}, \boldsymbol{Z}), \boldsymbol{y}\big) \tag{11}$$

5 Cycle Unnoticeability

In multigraph transaction data, laundering activity is characterised by recurring structural patterns, most notably short cycles. For an adversary, a perturbation that degrades the GNN is only useful if it also escapes a defender that inspects the graph for signs of tampering. Unlike in the image domain, where unnoticeability has a visual criterion, graphs offer no such criterion; unnoticeability must instead be defined through numerical measures on graph statistics [11]. We require that a perturbation leave the graph statistically close to the clean transaction graph, so that a statistical monitoring system would not register the change. We introduce a regularization term that constrains the perturbation to preserve the statistical distribution of short cycles, following the cycle-counting formulas of [9, 16]. We focus on cycle lengths two and three, which are tractable to count exactly and correspond to reciprocal transfers and triangular routing patterns that characterize common laundering schemes. The framework naturally extends to additional unnoticeability terms for other graph statistics.

To characterize cycle structure, we define the multigraph adjacency matrix $\boldsymbol{A} = \boldsymbol{B}_{\text{src}} \boldsymbol{B}_{\text{dst}}^\top$, where entry $\boldsymbol{A}_{ij}$ counts the number of directed edges from node v_i to node v_j. The diagonal entry $(\boldsymbol{A}^c)_{ii}$ then counts directed walks of length c that return to v_i, so $\text{Tr}(\boldsymbol{A}^c)/c$ gives the total number of directed cycles of length c in the graph. For $c = 2$ and $c = 3$, these coincide with true simple cycles. For $c \geq 4$, however, walks revisit nodes and edges, so $\text{Tr}(\boldsymbol{A}^c)$ overcounts distinct cycles. We therefore limit the regularizer to cycle lengths up to C, where $C = 3$ in practice.

The cycle regularizer measures the normalized mean squared error in cycle counts between the original and perturbed graph. Because the regularizer should penalize the addition or removal of cycles rather than the exact number of parallel edges forming them, we apply an elementwise saturation function $\phi \colon \mathbb{R} \to [0,1]$ to the adjacency matrices before raising them to the c-th power. During training we use $\phi = \tanh$, which is differentiable and saturates to 1 for large positive entries. For evaluation of the metric on a discrete graph we use $\phi(x) = \min(x, 1)$, which clips multi-edges to 1 exactly but is not differentiable. The cycle regularizer is then

$$\mathcal{L}_{\text{cycle}} = \frac{1}{n(C-1)} \sum_{c=2}^{C} \left(\frac{\text{Tr}(\phi(\boldsymbol{A})^c) - \text{Tr}(\phi(\boldsymbol{A}')^c)}{c} \right)^2, \tag{12}$$

where $\boldsymbol{A}' = \boldsymbol{B}'_{\text{src}} (\boldsymbol{B}'_{\text{dst}})^\top$ is the adjacency matrix of the perturbed graph. Minimizing $\mathcal{L}_{\text{cycle}}$ keeps the directed cycle profile of the perturbed graph close to the original.

Combining the cycle regularizer with the attack objective yields the final surrogate loss $\mathcal{L}' = \mathcal{L}_{\text{attack}} + \lambda_1 \mathcal{L}_{\text{cycle}}$, where $\mathcal{L}_{\text{attack}}$ is the differentiable surrogate loss (Appendix A) and $\lambda_1 \geq 0$ balances attack effectiveness against cycle unnoticeability.

6 Scalable Attacks and Adversarial Defenses

6.1 Scalable Attacks

The projected gradient descent (PGD) attack introduced in Section 4 can perform structural perturbations on multigraphs, but scales poorly with the graph size. The approach requires optimizing $O(m \cdot n)$ learnable parameters, which leads to infeasible memory demands for massive graphs encountered in real-world applications. To address this limitation, we adapt the *projected randomized block coordinate descent* (PR-BCD) method proposed by [5] to our incidence-matrix setting.

At each iteration t of the optimization loop previously described, we now operate on a restricted subset of edges $\mathcal{E}^{(t)} \subseteq \mathcal{E}$, and for each edge $e_j \in \mathcal{E}^{(t)}$, we further consider only a subset of nodes $\mathcal{V}_j^{(t)} \subseteq \mathcal{V}$. Let $b_\mathcal{E}$ and $b_\mathcal{V}$ denote the predetermined dimensions of this block of edges and nodes respectively. Typically, $b_\mathcal{E} \ll m$ and $b_\mathcal{V} \ll n$, and these block dimensions can remain fixed regardless of the overall graph size. This reduces the number of learnable parameters to $O(b_\mathcal{E} \cdot b_\mathcal{V})$. The relaxed incidence matrix $\hat{\boldsymbol{B}}$ from Section 4 is restricted to the active block: the $\boldsymbol{p}_j \boldsymbol{Q}_{ji}$ term applies only when $e_j \in \mathcal{E}^{(t)}$ and $v_i \in \mathcal{V}_j^{(t)}$, and is zero otherwise.

To ensure continued exploration of the perturbation space, we periodically refresh the active blocks. Whenever at least half of the edge-selection probabilities $\boldsymbol{p}$ are nonzero, we replace the 50% of edges with the smallest probabilities by newly sampled edges and target candidates. This block refresh prevents overfitting to the initial subset of edges and allows the attack to explore a broader region of the perturbation space. After a predetermined number of iterations, we stop resampling and continue optimizing the final block, after which we again sample discrete incidence matrices as previously described. The block size must satisfy $b_\mathcal{E} \geq \Delta$ to ensure that there are enough edges in the block to maximally utilize the rewiring budget.

6.2 Adversarial Training

To enhance the robustness of multigraph GNNs against structural perturbations, we incorporate adversarial training into the learning process. We approximate the inner maximization of the adversarial training objective from Section 3 using the PGD or PR-BCD attack methods described above, generating adversarial examples at each training step. To scale to large graphs, we construct mini-batches via neighborhood sampling [8]: at each training step, a set of target nodes is drawn and their K-hop subgraph is extracted; both the inner attack and the model update operate on this subgraph, keeping memory requirements independent of the global graph size. The rewiring budget Δ is applied relative to the edges present in the sampled subgraph. This approach encourages the model to learn parameters that are robust to worst-case perturbations within the specified budget. Because the examples do not need to be discrete, we directly employ the soft perturbed incidence matrix $\hat{\boldsymbol{B}}$ to compute the loss and update the model parameters, avoiding the overhead of sampling discrete instances.

7 Empirical Evaluation

We evaluate the framework along its three contributions: the structural attack on multigraph GNNs (Section 7.2), the cycle unnoticeability loss term (Section 7.3), and adversarial training as a defense (Section 7.4). Section 7.1 describes the shared experimental setup.

7.1 Experimental Setup

Dataset. We evaluate on the IBM synthetic multigraph benchmark used by [4], generated with the directed-multigraph pattern generator. Each split is an independently generated directed multigraph of 8192 nodes with an average node degree of 6, giving roughly 49,000 transaction edges per graph. Training, validation, and test graphs are generated from separate seeds, so the model never observes the test-graph structure during training and the setting is fully inductive. Every node carries a binary label for each of 11 structural patterns: in-degree, out-degree, fan-in, and fan-out, each positive when the corresponding count exceeds a fixed threshold; directed cycles of length two through six; scatter-gather; and biclique. These patterns mirror laundering typologies such as reciprocal transfers, triangular routing, fan-in and fan-out smurfing, and scatter-gather layering.

Metric. The pattern labels are heavily imbalanced, so we report the area under the precision-recall curve (PR-AUC) per pattern. The macro average is the unweighted mean of the 11 per-pattern PR-AUC values. PR-AUC is threshold-free and reflects detection of the rare positive nodes, unlike accuracy. For patterns where the positive class is the majority, we report PR-AUC for the negative class instead, so each per-pattern value always corresponds to the minority class.

Model. The detector is the per-edge multigraph GIN of [4], with reverse message passing, edge-feature updates, ego-node identifiers, and port numberings. We use 6 message-passing layers, hidden dimension 64, and ego dimension 32, with dropout 0.1 and a learnable ϵ. Port numbers are assigned dynamically within each mini-batch rather than globally, so the port vocabulary only needs to span the ports that occur in a batch; this keeps the port embedding small, at dimension 8 over a vocabulary of 32. Models train for 120 epochs with learning rate 10^{-3}, weight decay 10^{-4}, and a cosine-annealing-with-warm-restarts schedule ($T_0 = 8$, $T_{\text{mult}} = 2$, minimum learning rate 10^{-6}). Mini-batches are formed by neighborhood sampling [8] with 32 target nodes and a per-hop fan-out of $[10, 10, 10, 10, 5, 5]$ in both edge directions.

Attack protocol. All attacks use the PR-BCD attack of Section 6 on the incidence-matrix reformulation of Section 4, in the white-box evasion setting with fixed model parameters. Each attack runs for 100 optimization steps with 50 block resamples and learning rate 0.1, and selects the strongest of $K = 25$ sampled discrete perturbations. The block size defaults to $\sqrt{|\mathcal{V}|}$ candidate nodes and 2Δ edges. The rewiring budget Δ is set as an attack ratio of the edges in each sampled subgraph, swept over $\{0, 0.5, 1, 2, 5, 10\}$%. The non-learnable baseline, denoted *naive*, rewires the same number of edges to uniformly random targets and applies the identical $K = 25$ sampling. The cycle regularizer of Section 5 is disabled ($\lambda = 0$) for the attack-effectiveness experiments and swept separately in Section 7.3. Because PR-BCD operates on neighborhood-sampled subgraphs, its memory use is independent of the global graph size,

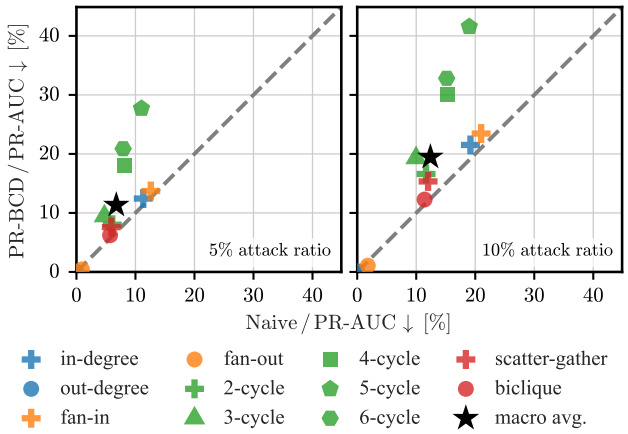

Figure 2: PR-AUC drop (percentage points) per pattern, comparing the learned PR-BCD attack (y-axis) with uniform random rewiring (x-axis), at a 5% (left) and 10% (right) rewiring budget. Each marker is one of the 11 structural patterns, with the macro average shown as a star. The dashed line is the diagonal $y = x$.

which is what makes attacking and adversarially training on these graphs feasible and what allows the framework to scale to larger networks.

Seeds and aggregation. Every reported point aggregates 9 runs, from 3 model-initialization seeds crossed with 3 attack seeds. Error bars and shaded bands denote one standard deviation across these runs. All experiments run on a single GPU.

7.2 Structural Attack Effectiveness

We first compare the learned PR-BCD attack against the non-learnable naive baseline at equal budget. Figure 2 plots the PR-AUC drop of each attack per pattern. Almost every pattern, and the macro average, lies above the diagonal at both the 5% and 10% budgets, so the learned attack removes more detection accuracy than uniform random rewiring. The gap is largest for the higher-order cycle patterns, which the learned attack disturbs while random rewiring rarely reaches.

To assess the plain robustness of the multigraph GIN on the subgraph-pattern dataset, we attack it with PR-BCD at increasing budgets. Figure 3 traces PR-AUC per pattern as the budget grows from 0% to 10%. The macro average falls from 99% on the clean graph to 80% at a 10% budget, and the degradation is uneven across patterns. Detection of 5- and 6-cycles drops the most, from near 100% to below 65%. Out-degree and fan-out instead stay near 100%: the attack rewires only edge targets, so a node's out-degree, the number of edges leaving it, never changes, and its fan-out, the number of distinct nodes it points to, changes only slightly and only for the few nodes whose outgoing edges are rewired.

7.3 Cycle Unnoticeability Trade-off

To quantify how the cycle unnoticeability term trades attack effectiveness against macro-level noticeability, we sweep its weight λ at a 10% budget. Figure 4 reports attack effectiveness as PR-AUC

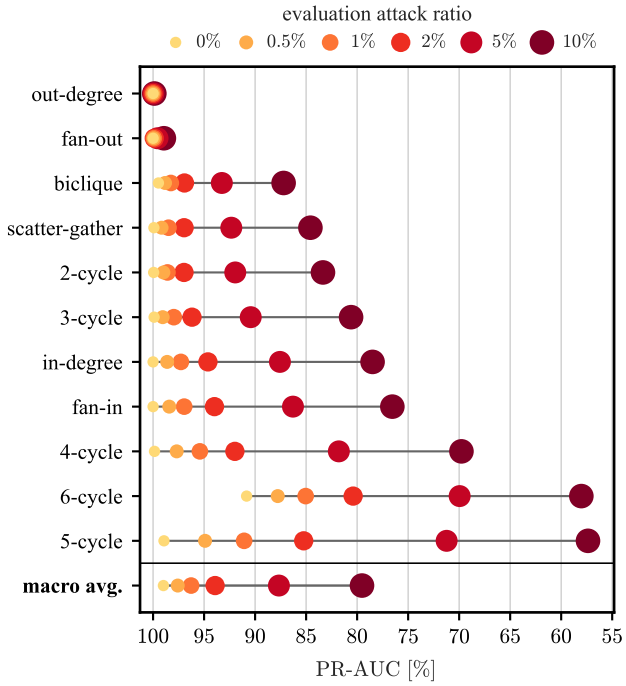

Figure 3: Per-pattern PR-AUC of the multigraph GIN under the PR-BCD attack as the rewiring budget increases from 0% to 10% of edges, with the macro average in the bottom row. Dot color and size encode the attack ratio, and each row spans its lowest-to-highest PR-AUC. The x-axis is reversed.

and noticeability as the square-root cycle-count error between the original and perturbed graph. Preserving the cycle-count distribution leaves the macro-level statistical fingerprint that a monitoring system inspects unchanged, so the attack cannot be flagged by a cycle-count monitor independent of the GNN [11].

Raising λ from 0 to 0.05 drives the cycle-count error down to its floor while the macro PR-AUC degradation stays essentially unchanged. Beyond 0.05 the cycle-count error is already saturated, and PR-AUC on the 2-cycle, 3-cycle, and macro-average tasks recovers as λ grows toward 1. The weight 0.05, marked in the figure, sits at the knee of the trade-off, and we use it for the adversarial-training experiments.

7.4 Adversarial Training Defense

To test whether adversarial training restores robustness to the structural attack, we train models with the procedure of Section 6 at several budgets and evaluate each under the PR-BCD attack. Figure 5 reports macro PR-AUC across test budgets from 0% to 10%. The undefended baseline falls from 99% on the clean graph to 80% under a 10% attack. Every adversarially trained model instead retains about 88% under the same attack while keeping clean accuracy near 99%, and the recovered robustness is largely insensitive to the training budget: training against a 0.5% attack already yields most of the benefit.

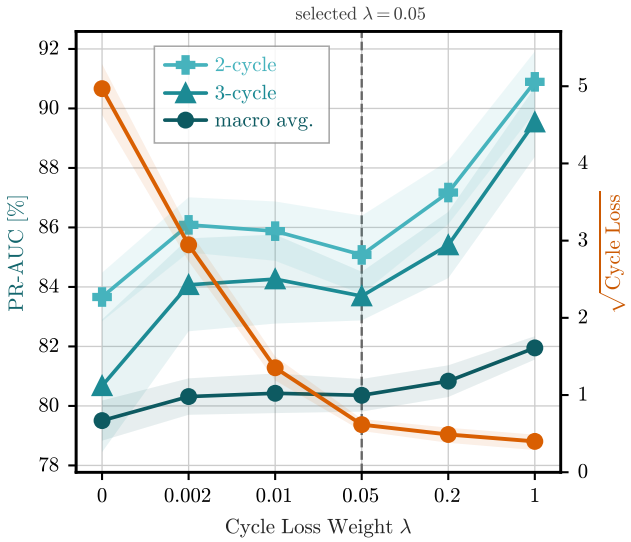

Figure 4: Attack effectiveness and macro-level noticeability as a function of the cycle unnoticeability weight λ, at a 10% rewiring budget. Left axis: PR-AUC for the 2-cycle, 3-cycle, and macro-average tasks. Right axis: square-root cycle-count error between the original and perturbed graph. Bands denote one standard deviation over 9 runs. The dashed line marks the selected $\lambda = 0.05$.

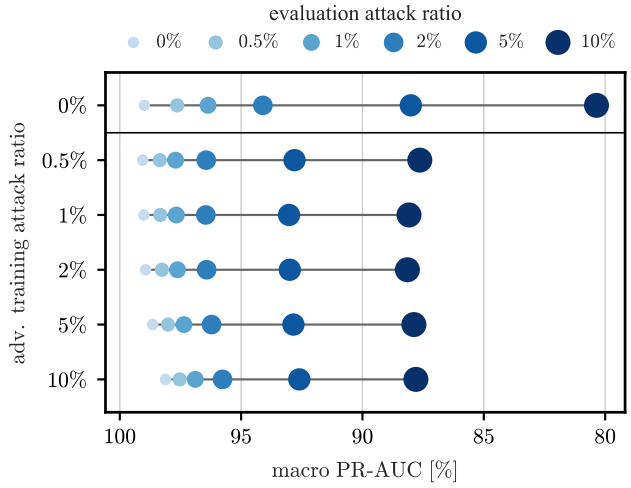

Figure 5: Macro PR-AUC of models trained with adversarial training at different budgets (rows; 0% is the undefended baseline, separated at the top), each evaluated under the PR-BCD attack at test budgets from 0% to 10%. Dot color and size encode the test attack ratio, and the x-axis is reversed.

8 Broader Impact

Money laundering is estimated to account for 2% to 5% of global GDP annually, cycling illicit proceeds back into the legitimate economy [15]. These funds sustain organized crime, drug trafficking, corruption, and terrorist financing; disrupting the laundering process therefore reduces the financial capacity of the criminal operations that depend on it. Graph neural networks trained on transaction networks offer a scalable, automated path toward this disruption, making adversarial robustness not merely a technical property but a prerequisite for meaningful real-world impact. A detector that can be systematically evaded provides only illusory protection, validating laundering schemes precisely when adversarial pressure is highest.

Our work introduces a systematic framework for generating structural adversarial attacks against multigraph GNNs on large financial networks. In financial crime detection, a successful attack means illicit transactions are classified as legitimate, enabling money laundering schemes to evade automated systems with direct financial and regulatory consequences. While the full-knowledge threat model limits immediate practical risk compared to black-box settings, it establishes an upper bound on vulnerability and provides the worst-case evaluation that practitioners and regulators need.

Our adversarial training defense recovers the accuracy lost under attack, as demonstrated in Section 7. Nevertheless, adversarial training introduces a robustness-accuracy trade-off: classification accuracy on clean graphs may decrease as the model learns to resist perturbations. Practitioners should evaluate this trade-off for their specific deployment context and tune the unnoticeability constraints to match their graph's distributional properties. Both false negative rates for missed laundering and false positive rates for blocked legitimate transactions should be monitored under clean and adversarial conditions.

9 Conclusion and Discussion

This work presents the first structural attack and adversarial training framework for per-edge multigraph GNNs, whose robustness has been unstudied despite their growing role in financial crime detection. The incidence-matrix reformulation of multigraph GNN message passing enables the first gradient-based structural attack on per-edge multigraph GNNs. Learned attacks cause multigraph GNNs to mispredict money laundering cycle patterns and substantially outperform uniform random rewiring at the same budget, confirming that the framework exploits structural vulnerabilities that non-learned perturbations miss. Cycle unnoticeability regularization provides meaningful control over the realism–effectiveness trade-off: modest regularization substantially reduces the structural noticeability of the attack while leaving PR-AUC degradation nearly unchanged, but beyond a critical weight attack effectiveness degrades sharply. Scaling via projected randomized block coordinate descent makes the framework memory-efficient for large financial networks, and adversarial training on perturbed examples hardens multigraph GNNs against structural attacks, enabling both evaluation and defense within a single framework.

The framework targets structural perturbations only, specifically edge rewiring. Node and edge feature perturbations fall outside the

current scope. Feature perturbations are a natural extension, as the same PGD-based optimization applies directly once node and edge features are treated as continuous.

Evaluation uses a graph pattern detection benchmark rather than a direct AML classification dataset with licit/illicit transaction labels. Pattern detection exercises the same structural vulnerabilities that matter in the AML setting, so the results suggest the framework transfers to transaction-level AML data; confirming this is a natural next step. More robust AML models would directly benefit financial institutions, where detection systems that resist adversarial manipulation reduce the ability of financial criminals to evade automated monitoring.

The cycle regularizer enforces rule-based unnoticeability: it encodes domain knowledge that short-cycle distributions should be preserved under perturbation, constraining this through a fixed loss term. Further statistics can be incorporated through additional rule-based terms. A learned unnoticeability constraint would reduce the reliance on domain knowledge by replacing these fixed statistics with a model that scores edge plausibility from the data directly.

Future work should extend the framework to node and edge feature perturbations, as incorporating edge features can make attacks more realistic and effective. The cycle regularizer demonstrates that a single topological statistic can meaningfully constrain attack noticeability; a natural extension replaces this fixed statistic with a learned model that scores how plausible each edge is given the graph and its labels, guiding the attack toward perturbations that remain undetectable to a data-driven monitor. Adapting such a model to multigraphs, where parallel edges require per-edge rather than per-pair plausibility scoring, remains an open direction.

References

[1] Andersen Ang, Jianzhu Ma, Nianjun Liu, Kun Huang, and Yijie Wang. 2021. Fast projection onto the capped simplex with applications to sparse regression in bioinformatics. In *NeurIPS*.

[2] H. Çağrı Bilgi, Lydia Y. Chen, and Kubilay Atasu. 2024. Multigraph Message Passing with Bi-Directional Multi-Edge Aggregations. *arXiv:2412.00241* (2024).

[3] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc Von Niederhäusern, Erik Altman, Haris Pozidis, and Kubilay Atasu. 2024. Graph feature preprocessor: Real-time subgraph-based feature extraction for financial crime detection. In *ACM ICAIF*.

[4] Béni Egressy, Luc Von Niederhäusern, Jovan Blanuša, Erik Altman, Roger Wattenhofer, and Kubilay Atasu. 2024. Provably powerful graph neural networks for directed multigraphs. In *AAAI*.

[5] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. 2021. Robustness of graph neural networks at scale. In *NeurIPS*.

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *ICLR*.

[7] Lukas Gosch, Simon Geisler, Daniel Sturm, Bertrand Charpentier, Daniel Zügner, and Stephan Günnemann. 2023. Adversarial training for graph neural networks: Pitfalls, solutions, and new directions. In *NeurIPS*.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*.

[9] Frank Harary and Bennet Manvel. 1971. On the number of cycles in a graph. *Matematickỳ časopis* (1971).

[10] Rasmus Ingemann Tuffveson Jensen, Joras Ferwerda, and Christian Remi Wewer. 2025. Searching for smurfs: Testing if money launderers know alert thresholds. *Journal of Quantitative Criminology* (2025).

[11] Hyeonsoo Jo, Hyunjin Hwang, Fanchen Bu, Soo Yong Lee, Chanyoung Park, and Kijung Shin. 2025. On measuring unnoticeability of graph adversarial attacks: observations, new measure, and applications. In *KDD*.

[12] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.

[13] Konstantinos Sotiropoulos, Lingxiao Zhao, Pierre Jinghong Liang, and Leman Akoglu. 2023. Adamm: Anomaly detection of attributed multi-graphs with metadata: A unified neural network approach. In *BigData*.

[14] Linlin Su, Zeming Gan, Jinyan Wang, and Xianxian Li. 2025. Structural Complementary Hypergraph Defense Framework against Adversarial Attacks. *Knowledge-Based Systems* (2025).

[15] United Nations Office on Drugs and Crime (UNODC). 2011. *Estimating Illicit Financial Flows Resulting from Drug Trafficking and Other Transnational Organized Crimes.* Technical Report. United Nations.

[16] A. N. Voropaev and S. N. Perepechko. 2012. The Number of Fixed Length Cycles in Undirected Graph Explicit Formula in Case of Small Lengths. *Discrete and Continuous Models and Applied Computational Science* (2012).

[17] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology attack and defense for graph neural networks: An optimization perspective. In *IJCAI*.

[18] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.

[19] He Zhao, Zhiwei Zeng, Yongwei Wang, Deheng Ye, and Chunyan Miao. 2024. HGAttack: Transferable Heterogeneous Graph Adversarial Attack. *arXiv:2401.09945* (2024).

[20] Tianzhe Zhao, Jiaoyan Chen, Yanchi Ru, Qika Lin, Yuxia Geng, and Jun Liu. 2024. Untargeted Adversarial Attack on Knowledge Graph Embeddings. In *ACM SIGIR*.

A Multilabel Surrogate Loss

The accuracy-based loss $\mathcal{L}_{0/1}$ in the attack objective is non-differentiable and cannot be optimized directly with gradient-based methods. Following [5], we replace it with a differentiable surrogate. The surrogate losses in [5] assume mutually exclusive classes and do not generalize to multilabel settings, such as the benchmarks of [4], where nodes carry multiple labels. We therefore introduce a new surrogate that handles multilabel outputs directly.

Let $z_{ic} \in \mathbb{R}$ be the raw logit for node $i \in \{1, \ldots, N\}$ and class $c \in \{1, \ldots, C\}$, and let $y_{ic} \in \{0, 1\}$ be the ground-truth label. We map each binary label to a signed value $2y_{ic} - 1 \in \{-1, +1\}$ and define the attack loss as

$$\mathcal{L}_{\text{attack}} = \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} \sigma\left((2y_{ic} - 1)\, z_{ic}\right), \tag{A.1}$$

where σ is the sigmoid function. Correctly classified nodes produce a positive argument $(2y_{ic} - 1)z_{ic} > 0$, pushing $\sigma(\cdot)$ toward one. Misclassified nodes produce a negative argument, pushing $\sigma(\cdot)$ toward zero. In both extremes the gradient of σ is near zero, so confidently classified or confidently misclassified nodes contribute little to the gradient. Gradient signal therefore concentrates on nodes with logits near zero, i.e., near the decision boundary, where small structural perturbations are most likely to flip predictions. The attack thus allocates its perturbation budget where it is most effective.

References

[1] Erik Altman et al. “Realistic synthetic financial transactions for anti-money laundering models”. In: *NeurIPS*. 2023.

[2] Andersen Ang et al. “Fast projection onto the capped simplex with applications to sparse regression in bioinformatics”. In: *NeurIPS*. 2021.

[3] H. Çağrı Bilgi, Lydia Y. Chen, and Kubilay Atasu. “Multigraph Message Passing with Bi-Directional Multi-Edge Aggregations”. In: *arXiv:2412.00241* (2024).

[4] Jovan Blanuša et al. “Graph feature preprocessor: Real-time subgraph-based feature extraction for financial crime detection”. In: *ACM ICAIF*. 2024.

[5] Béni Egressy et al. “Provably powerful graph neural networks for directed multigraphs”. In: *AAAI*. 2024.

[6] Ron-Marco Friedrich and Franz Faupel. “Adaptive Model for Magnetic Particle Mapping Using Magnetoelectric Sensors”. In: *Sensors* 22.3 (2022), p. 894. DOI: 10.3390/s22030894.

[7] Simon Geisler et al. “Robustness of graph neural networks at scale”. In: *NeurIPS*. 2021.

[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *ICLR*. 2015.

[9] Lukas Gosch et al. “Adversarial training for graph neural networks: Pitfalls, solutions, and new directions”. In: *NeurIPS*. 2023.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *NeurIPS*. 2017.

[11] Rasmus Ingemann Tuffveson Jensen, Joras Ferwerda, and Christian Remi Wewer. “Searching for smurfs: Testing if money launderers know alert thresholds”. In: *Journal of Quantitative Criminology* (2025).

[12] Hyeonsoo Jo et al. “On measuring unnoticeability of graph adversarial attacks: observations, new measure, and applications”. In: *KDD*. 2025.

[13] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations*. 2017.

[14] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *International Conference on Learning Representations*. 2018.

[15] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[16] United Nations Office on Drugs and Crime (UNODC). *Estimating Illicit Financial Flows Resulting from Drug Trafficking and Other Transnational Organized Crimes*. Tech. rep. United Nations, 2011.

[17] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018.

[18] Boris Weisfeiler and A. A. Lehman. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16.

[19] Kaidi Xu et al. “Topology attack and defense for graph neural networks: An optimization perspective”. In: *IJCAI*. 2019.

[20] Keyulu Xu et al. “How powerful are graph neural networks?” In: *ICLR*. 2019.

[21] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. “Adversarial attacks on neural networks for graph data”. In: *KDD*. 2018.

A

Proofs

A.1. Adjacency Matrix Factorization

Proof that $\boldsymbol{A} = \boldsymbol{B}_{src}\boldsymbol{B}_{dst}^{\top}$. Fix any pair of nodes $v_i, v_j \in \mathcal{V}$. By definition, $(\boldsymbol{B}_{\text{src}})_{ik} = 1$ if edge e_k leaves v_i and 0 otherwise, and $(\boldsymbol{B}_{\text{dst}})_{jk} = 1$ if edge e_k enters v_j and 0 otherwise. By the definition of matrix multiplication, $(CD)_{ij} = \sum_k C_{ik}D_{kj}$. Substituting $C = \boldsymbol{B}_{\text{src}}$ and $D = \boldsymbol{B}_{\text{dst}}^{\top}$, and applying the transpose identity $(\boldsymbol{B}_{\text{dst}}^{\top})_{kj} = (\boldsymbol{B}_{\text{dst}})_{jk}$, the (i, j) entry of the product is

$$(\boldsymbol{B}_{\text{src}}\boldsymbol{B}_{\text{dst}}^{\top})_{ij} = \sum_{k=1}^{m} (\boldsymbol{B}_{\text{src}})_{ik}\,(\boldsymbol{B}_{\text{dst}})_{jk}. \tag{A.1}$$

Because both factors are $\{0, 1\}$-valued, the k-th term equals 1 if and only if e_k both leaves v_i and enters v_j, i.e., $e_k = (v_i, v_j)$. The sum therefore counts the number of directed edges from v_i to v_j, which is $\boldsymbol{A}_{ij}$ by definition. Since (i, j) was arbitrary, $\boldsymbol{A} = \boldsymbol{B}_{\text{src}}\boldsymbol{B}_{\text{dst}}^{\top}$. □